XML Schema Component Re-use

Cheryl Connors
Mike Cokus
Ed Masek
Dr. Bob Miller
Dan Winkowski

MITRE
Information Interoperability

Contents

- Background
- Re-use and Interoperability
- A short review of XML Schemas
- Configuration Management
- Re-use Components
- XML Schema Capabilities for Re-use
- Schema Design for Re-use
- Lessons Learned
- Summary

Personal Introduction

- **Education**
  - BS in Computer Science, MS in Information Systems
- **Background**
  - VMF and MTF Messaging
  - Prototype Database Development for Army and Airforce
- **XML and XML Schemas**
  - Supported XML-MTF Development effort
  - Lead for the XML-VMF Development Team
    - XML-VMF Mapping Specification (ICP awaiting approval)
    - XML-VMF Schema Specification
  - Teamed with LM to develop a “Common” Schema for an XML-VMF/JCDB Operations Order
  - TADIL Gateway MOIE
    - Designed schema format for Link-16
    - Designed schema format for ADS-B (Civilian)
  - Current effort is to support AODB Schema development

Background on the Re-use Effort

- Leveraging MITRE D540 expertise on XML-MTF to assist Family of Interoperable Operational Pictures (FIOP)
  - FIOP wanted to re-use XML-MTF Data Elements
  - Have to register non MTF-XML Data Elements
  - LM plans to move from DTD to Schema eventually
- D540 participation:
  - Assist correlating the Air Battle Plan (ABP) elements to semantically equivalent USMTF Data Elements
  - Compare FIOP and USMTF data definitions for discrepancies
  - Develop an ABP Schema using the correlated XML-MTF XML elements and type definitions
- Lessons Learned should be applicable to similar efforts
- Does not address code re-use for XML objects

General Re-use Applicability

- Re-use broadly applicable: XML is just one way to re-use information (info) objects
- Re-use can be applied to:
  - Semantics
    - Describes an object in the context of how it's used
  - Syntactic representations
    - Describes the form, structure, symbols, tokens, types and order of the information (e.g.,"string, 8 chars")
  - Documentation
    - Provides additional information
  - Programming code
  - Stylesheets
  - etc.
- Re-use is valuable because it will:
  - Save development time and dollars
  - Improve chances of interoperability

Re-use and Interoperability

- In a **perfect** world, an info object would be defined once (in context) and all communities would re-use that definition to get **interoperability**
- Will this ever happen?
  - Remember the US's conversion to the Metric system?
  - Wouldn't it be good to have a standard set of measurements with the rest of the world?

- **Example: In a well-known mishap, the Space Shuttle was conducting a test of the Star Wars laser system, with a laser beam sent from Hawaii. Because of human error (*converting miles to meters incorrectly*), the Shuttle was oriented in space away from Hawaii, not towards it, and missed the signal completely.**

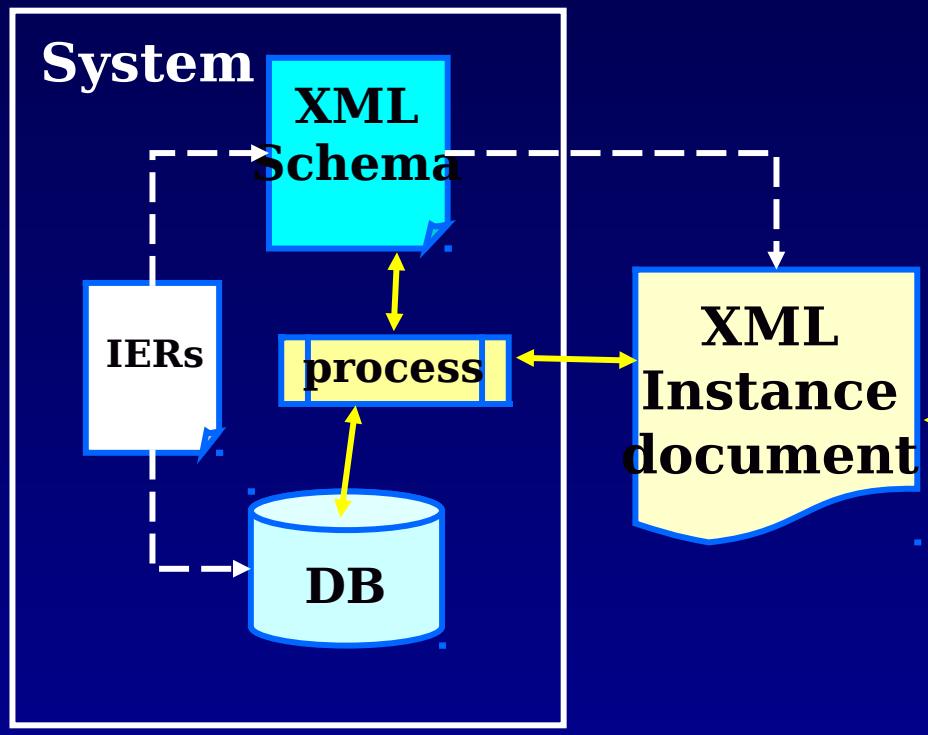
Conclusion: Would avoid similar translation errors if we all used the same standard unit of measurements.

Re-use and Interoperability (Cont.)

- Back in the real world, we don't all have the same info objects. Why not?
  - One size doesn't fit all
  - Can't always afford to retrofit, re-program, re-train, etc.,
- However, associated domains will benefit from re-using a defined set of common info. objects
  - DoD XML Registry provides a method to store the definitions and allow them to be re-used
- XML and XML Schemas provide strong support for re-use

A short review of XML Schemas

- **XML Schemas:**
  - Describe the valid format and contents of an XML Instance document (e.g., cardinality, conditionality, repeatability etc.,)
- **XML Instance documents:**
  - Contain tagged information



XML Schemas validate the XML Instance documents. Benefit: Avoids need to develop a validation parser

TX/RX Instance documents between systems, to a web server, etc.

A short review of XML Schemas (cont.)

XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
…
<element name= “revision _dtg”>
  <complexType>
    <sequence>
      <element name=“year ”
type=“year.2.ch”
                minOccurs =“1”/>
      <element name=“month ”
type=“alpha.month”
                minOccurs =“1”/>
          etc.
      </restriction >
    </ sequence>
  </complexType>
</element>
<simpleType name= “year.2.ch”>
  <restriction base=“integer”>
    <minInclusive value=“1945”/>
    <maxInclusive value=“2002”/>
 </restriction>
</simpleType>
</schema>
```

XML Instance Document

```
<?xml version="1.0” etc. >
<msg:operations_order  … etc
…
<revision_dtg>
      <year>2001</year>
      <month>MAY</month>
      <day>14</day>
      <hour>2</hour>
      <minute>15</minute>
</revision_dtg>
…
</ msg:operations_order>
```

Configuration Management (CM)

- Many issues about CM of XML and any re-used components that need to be resolved
- Can't physically reach the original Schemas or components available at the DoD XML Registry
  - Will most likely make local copies
    - How format re-used components? Maintain original construction like XML-MTF? Alter it? (e.g., package XML Elements as complete objects?)
  - How keep in synch with original Schemas?
    - Register your usage with Registry to get updates? (Not a current capability)
  - Versioning
    - Have different versions in different namespaces?
  - How document/track any changes?
  - Etc.

Conclusion: Need to consider the CM of XML Schemas and re-used components

XML Schema Main Re-usable Components

- Schema
- XML Element
  - Defined in terms of one or more XML Elements, complexTypes, or simpleTypes
- ComplexType
  - Defined in terms of one or more XML Elements, complexTypes, or simpleTypes
- SimpleType[1]
  - Defined as a “simple” or “base” type, i.e., a “string”, “float”, etc.
  - Can be a union of named simpleTypes
- Attribute[1]
  - Must be defined as a simpleType
- Annotation (documentation)

Note 1: simpleTypes and Attributes may be defined by a “user-def

XML Element Construction

- **Element declarations are in terms of a type definition**
  - **simpleType or a collection of other Elements or types using the complexType definition**
- **An XML Element may contain an in-line definition:**

```
<element name="Count_of_Aircraft" type="xsd:integer"/>
```

- **Or an XML Element can be defined by a type definition that is located "outside" itself:**

```
<element name=" Count_of_Aircraft "
    type="SomeDB:Count_Of_Items"/>
```

This example uses a Namespace qualifier to link to a different XML Schema

SomeDB Schema

```
<simpleTypename="Count_Of_Items">
    <restriction base="xsd:integer">
        <minInclusivevalue="1"/>
        <maxInclusivevalue="999">
    </restriction>
</simpleType>
```

XML Elements and Attributes

- Debate on the distinction between them
  - One COI's Elements might another COI's Attributes
- No rules to distinguish between them
- Elements are more flexible than Attributes
  - Can be defined as complex or simple types
    - A complexType can be a collection (e.g., "sequence", "all"., etc.) of Elements or types
    - A simpleType must be defined as an XML base type like "string", "float" etc., or in terms of another simpleType that is an XML base type
  - Attributes
    - Must be defined as simpleTypes
    - Can't directly Attribute an Attribute

Conclusion: Since Attributes have so many restri they should be used only for meta-data

XML Element and Type Definitions

- **XML Elements aren't as flexible as XML type definitions. For example, assume:**

  <Element name= "Count_Of_Aircraft" type="xsd:integer"/>

- **Another Element *can't* re-use this definition directly:**

  <Element name= "Aircraft_Quantity" type="Count_Of_Aircraft"/>

  is ILLEGAL

- **Can re-use type definitions**

<Element name= "Count_Of_Aircraft" type="Count"/>

<Element name= "Aircraft_Quantity" type="Count"/>

```
<simpleTypename="Count">
    <restriction base="xsd:integer">
        <minInclusivevalue="1"/>
        <maxInclusivevalue="999">
    </restriction>
</simpleType>
```

XML Element and Type Definitions (Cont.)

- **Re-use of the simpleType definition level rather than at the XML Element Level has consequences:**
  - **Lose the knowledge that there is an equivalency between the XML Elements.**
  - **Also lose any of the original Element's semantic information (Annotation)**
  - **However, the ease of re-use makes type definitions more flexible and re-usable by far**
  - **Retains any semantic meaning (e.g., any additional explanations) that might be available at the type definition level**

Conclusion: Don't create Elements with in-line definitions. Instead create type definition outsi XML Element to make these definitions availab re-use.

Re-use Methods

- XML's Import
- XML's Include
- XML's ref
- XML's Substitution Groups
- XML's Attribute Groups
- XML's Restriction
- XML's Extension
- XML's Model and Attribute Group Definitions
- XML's Redefine
- etc.
- Cut and Paste
  - Did not use all of these capabilities. Will be discussing those capabilities we've utilized

Schema Capabilities for Re-use

- XML “Import”
  - Link to other namespaces (can be another Schema). Acts like a pointer to a specific information object(s) in the linked namespace(s)
  - Maintains semantic information of original information
  - Maintains meta-data about source and versioning
  - In future, might be possible to have changes to the original schema roll-over to re-used components (CM issue)
- XML“Include”
  - Creates one logical Schema from multiple namespaces (can be other Schema(s))
  - Same advantages as “Import”
  - If resultant Schema is very large, might impact processing speed

XML Schema Capabilities for Re-use

- XML's "import" or "include" (continued)
  - Can re-use Global elements and type definitions
  - Any elements and type definitions that exist as part of another definitions can't be re-used directly

```
<element>
    <element/>
    <element/>
    <element/>
</element>
```

These elements can't be pulled out individually and be re-used

- XML's "ref"
  - Can re-use Global elements
    - <element ref="Element_Name" minOccurs="1"/>
  - Can't change the XML Element's name
  - Can't change the XML Element's composition
  - Only "minOccurs", "maxOccurs", "id", and "annotation" allowed

Schema Capabilities for Re-use

- XML's Substitution Groups
  - Can modify the original XML Element: change name, restrict or extend the definition
  - Can have multiple members
  - Can't change the type (e.g., can't change integer to float)
  - Limited to Global XML Elements

- XML's "extension base"
  - "Adds" the original definition to the new element
  - Can't alter the original definition
  - Can't use XML elements to extend other elements

Schema Capabilities for Re-use (Cont.)

- XML's "extension base" Example:

```
<xsd:complexType name="base.type">
    <xsd:sequence>
        <xsd:element name="free_text" type="free.text" minOccurs="0"/>
        <xsd:element name="information" type="information"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="airspace.shape">
   <xsd:complexContent>
       <xsd:extension base="base.type">
           <xsd:sequence>
          <xsd:element name="track_info" nillable="true">
              <xsd:complexType>

                    ...
```

Result: the complexType "airspace.shape" contains the two elements defined in the complexType "base.type"

Schema Capabilities for Re-use (Cont.)

- **XML's "restriction base"**
  - **Can add restrictions to the original definition**
  - **In XML-MTF, most common usage of restriction was to specify legal values and ranges; for example:**

```
<xsd:simpleType name="latitude.degrees.17.1">
        <xsd:restriction base="xsd:integer">
        <xsd:minInclusive value="00"/>
        <xsd:maxInclusive value="90"/>
        <xsd:pattern value="[0-9]{2}"/>
</xsd:restriction>
</xsd:simpleType>
```

Other Capabilities for Re-use

- **Cut and paste**
  - **Just copy a registered XML component (element, type definition, etc.,) and paste into new Schema**
- **Consequences:**
  - **Lose logical link that identifies the original namespace of the information**
  - **Lose versioning information (if any)**
  - **Won't be updated when original schema is revised (if the CM process does someday supports this function)**

Conclusion: The XML "Import/re-use" capabilities are preferable to the "cut and paste" method to access re-usable components located in other namespaces

Cut and Paste Re-use Method:Extend

- This is a “Cut and paste” of an existing definition
  - Creates a super-set
  - CM issues: Because keeping same name, maintain “logical link” to original definition
  - However, there’s a disconnect between Registered and new definition

Extended copy of the type *(Located in the Local Namespace)*

```
<simpleType name= “aircraft.list”>
    <restriction base=“string">
        <enumeration value=“F4”/>
        <enumeration value=“F15”/>
        <enumeration value=“F16”/>
                    …
        <!-- total of 80 aircraft -->
                    …
        <enumeration value=“F-89 Zero”/>
    </restriction >
</ simpleType >
```

Registered XML Component *(Located in the XML-MTF Namespace)*

```
<simpleType name= “aircraft.list”>
    <restriction base=“string">
        <enumeration value=“F16”/>
                    …
    <!-- ONLY 10 aircraft listed -->
                    …
        <enumeration value=“TOE3”
    </restriction >
</ simpleType >
```

Cut and Paste Re-use Method: Restrict

- Another “Cut and paste” of an existing definition
  - Creates a sub-set
  - CM issues: Because keeping same name, maintain “logical link” to original definition
  - However, there’s a disconnect between Registered and new definition

Restricted copy of the type
(Located in the Local Namespace)

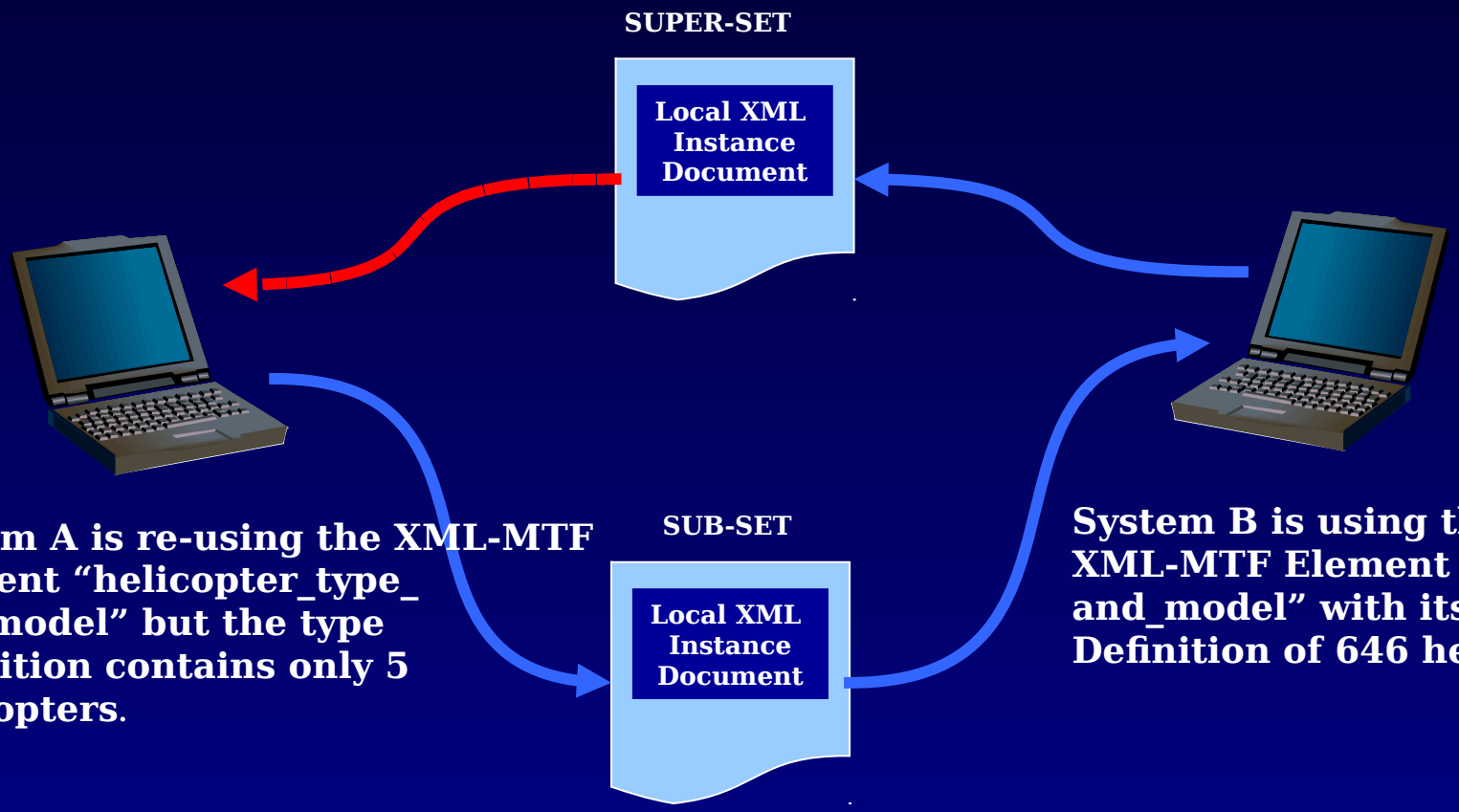
```
<simpleType name= “aircraft.list”>
        <restriction base=“string">
            <enumeration value=“F16”/>
                            …
            <!-- ONLY 10 aircraft listed -->
                            …
            <enumeration value=“TOE3”/>
        </restriction >
    </ simpleType >
```

Registered XML Component
(Located in the XML-MTF Namespace)

```
<simpleType name= “aircraft.list”>
        <restriction base=“string">
            <enumeration value=“F4”/>
            <enumeration value=“F15”/>
            <enumeration value=“F16”/>
                            …
            <!-- total of 4,000 aircraft -->
                            …
            <enumeration value=“Zero”/>
        </restriction >
    </ simpleType >
```

Impact of Restricting and Extending

e "Super -set" to the "Sub-set": May be INVALID for the system using th


SUPER-SET
Local XML Instance Document
SUB-SET
Local XML Instance Document
System A is re-using the XML-MTF Element "helicopter_type_ and_model" but the type Definition contains only 5 helicopters.
System B is using the registe XML-MTF Element "helicopt and_model" with its type Definition of 646 helicopters

From the "Sub-set" to the "Super-set": Will be valid for both end systen

Summary of Re-use Methodologies

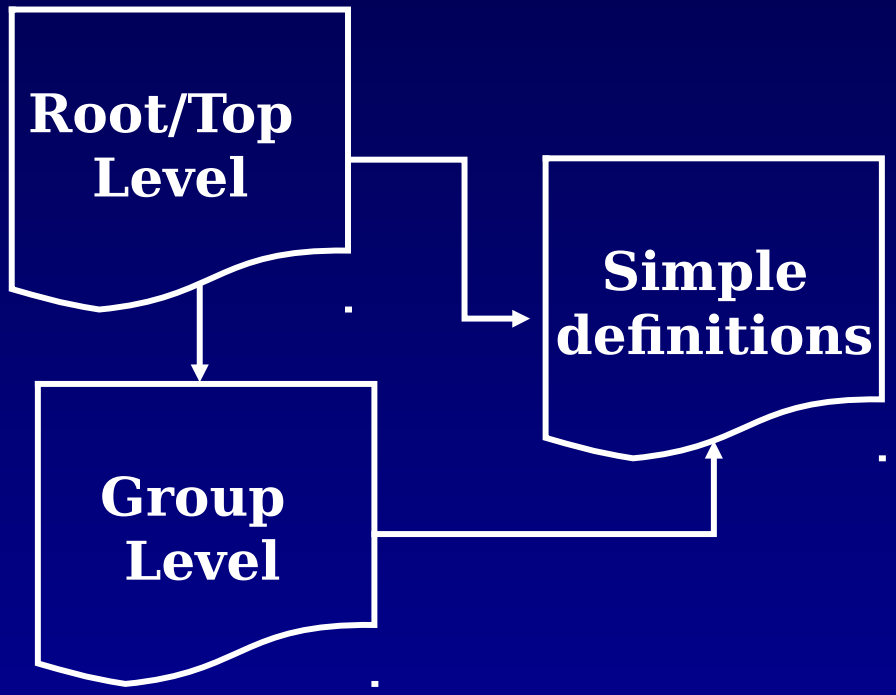
- XML provides many capabilities that support re-use
  - Many depend on using global definitions
  - Each re-use method has pros and cons
  - Elements are not as flexible to re-use as type definitions
- Other re-use methodologies (cut and paste) will be used
  - May have an impact on interoperability
- CM of the re-used Schema components is going to be an issue
- Thinking re-use should impact your Schema design

Schema Design for Re-use

- Should consider designing to maximize re-usability to:
  - Take advantage of registered XML components
  - So that others can re-use your new definitions
- No one Schema design will work for every source
- Consider the utility of creating Global XML Elements and Types in your Schema
  - Will have an impact on your design
  - Would select global types over global Elements
- Include all possible documentation
- Avoid in-line definitions
  - Not re-usable
  - Want to avoid defining the same information more than once, because data redundancy is bad

Schema Design for Re-use (cont.)

- An example Schema Structure design:
  - Top-level Schema with root element(s) and structural information
  - Another Schema for all the complexTypes
  - One Schema for all simpleTypes



Schema Design for Re-use (cont.)

- Should devise a rule set for determining component Schema location
  - simpleTypes pretty easy to figure out
  - complexTypes rules may be more difficult
- Why not just one Schema?
  - If it makes sense for your data, why not?
  - However, suggest logical “sections” for each component type (elements, types)
- From experience, having at least a separate Schema for the simpleTypes is helpful
- Bottom line: No matter the final design, develop a documented, authoritative plan that anyone using your data source can use for Schema development

Other Lessons Learned

- **Define a standard Naming Convention and enforce it**
  - **Whatever one you use, be consistent**
- **Remove yourself from the design**
  - **Use the existing information, don't add or subtract to it**
    - **For example, spelling out acronyms for the tag names; if the database field or message standard doesn't spell it out, neither should you.**
- **Create a Mapping and Schema Specification**
- **These suggestions will help:**
  - **Avoid having an internally inconsistent Schema design**
  - **Avoid having different developers develop non-interoperable XML Instance documents**
  - **Make it easier to automate the Schema derivation**
  - **Easier to maintain**

Summary

- Task was to re-use registered XML-MTF Data Elements
- Concluded that there are various mechanisms that exist for re-use
- Each type of re-use has pros and cons
- Type definitions appear to be more flexible for re-use than XML Elements
- A standard Naming Convention is needed
- Concluded that re-use should be considered during the design of the Schema
- Having a documented design plan (Mapping and Schema Specification) for Schemas is important